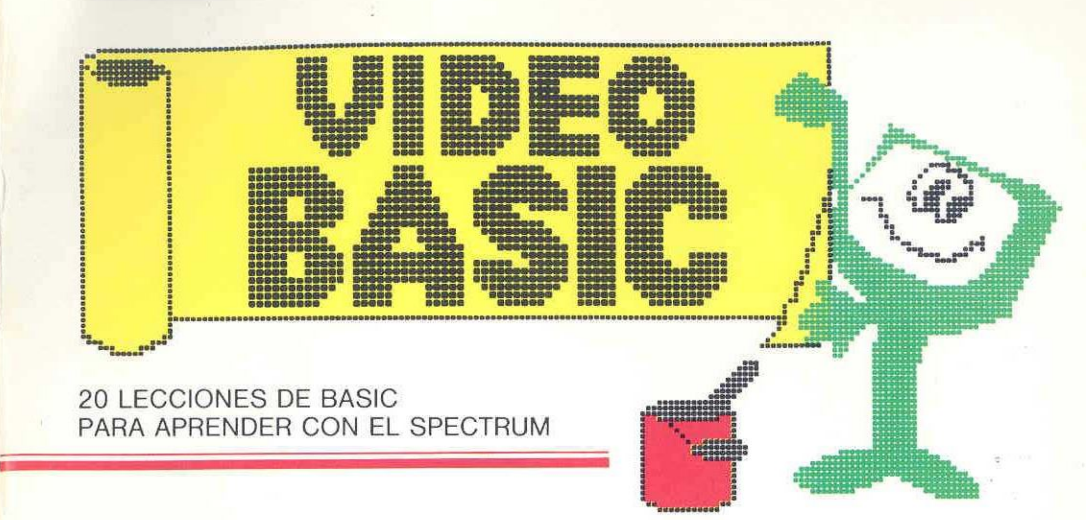


20 LECCIONES DE BASIC
PARA APRENDER CON EL SPECTRUM

INGELEK



JACKSON

- CPU: organización externa e interna
- Los BUSES
- El código máquina
- Ventajas y desventajas del Assembler
- USR
- Memorización de programas en código máquina
- Videoejercicios
- Videojuego N.º 18

18 Spectrum

16K/48K/PLUS

VIDEO BASIC
Una publicación de
INGELEK JACKSON
Director editor por INGELEK:
Antonio M. Ferrer
Director editor por JACKSON HISPANIA:
Lorenzo Bertagnolio
Director de producción:
Vicente Robles
Autor: Softidea
Redacción software italiano:
Francesço Franceschini,
Stefano Cremonesi
Redacción software castellano:
Fernando López, Antonio Carvajal,
Alberto Caffarato, Pilar Manzanera
Diseño gráfico:
Studio Nuovaidea
Ilustraciones:
Cinzia Ferrari, Silvano Scolari,
Equipo Galata
Ediciones INGELEK, S. A.
Dirección, redacción y administración,
números atrasados y suscripciones:
Avda. Alfonso XIII, 141
28016 Madrid. Tel. 2505820
Fotocomposición: Espacio y Punto, S. A.
Imprime: Gráficas Reunidas, S. A.

ISBN del tomo 4: 84-85831-20-9
ISBN del fascículo: 84-85831-11-X
ISBN de la obra completa: 84-85831-10-1
Depósito Legal: M-15076-1985
Plan general de la obra:
20 fascículos y 20 casetes, de aparición quincenal, coleccionables en 5 estuches.
Distribución en España:
COEDIS, S. A.
Valencia, 245. 08007 Barcelona.
INGELEK JACKSON garantiza la publicación de todos los fascículos y casetes que componen esta obra y el suministro de cualquier número atrasado o estuche mientras dure la publicación y hasta un año después de terminada.
El editor se reserva el derecho de modificar el precio de venta del fascículo, en el transcurso de la obra, si las circunstancias del mercado así lo exigen.
Diciembre, 1985
Impreso en España.

INGELEK



JACKSON

SUMARIO

Introducción

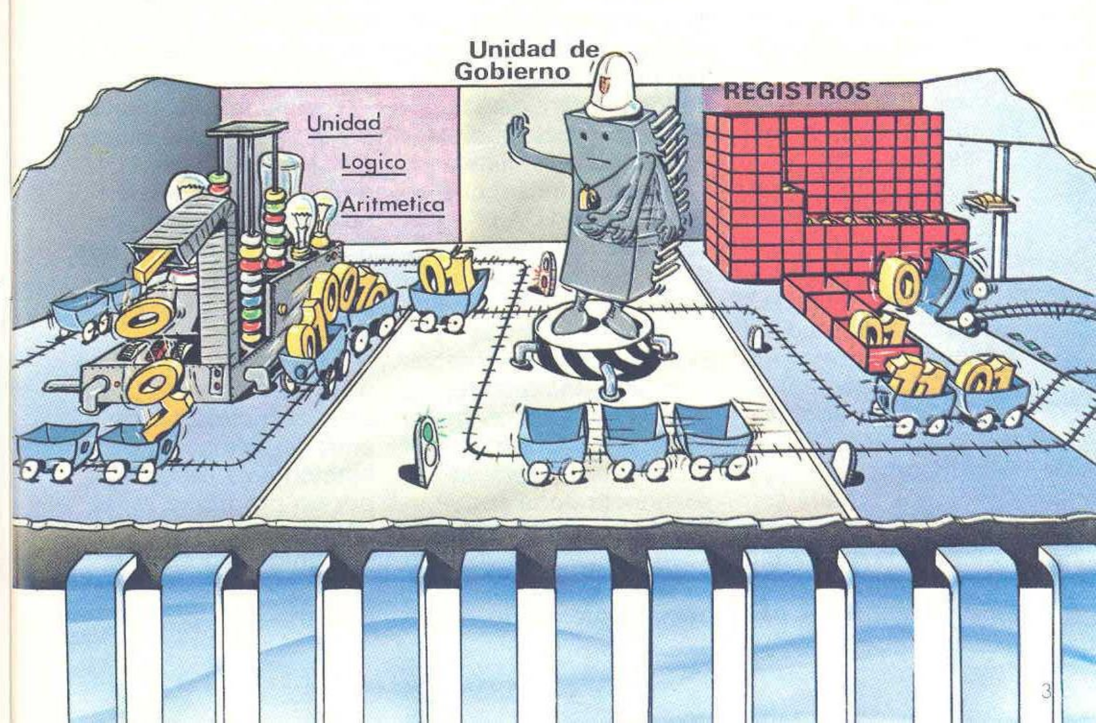
Antes de continuar... demos algunos pasos hacia atrás. Para presentar el temido código máquina (C/M para quien quiera presumir) y el igualmente conocido assembler, es necesario refrescar los conceptos base de la CPU, su organización externa e interna, los buses.
No se trata de un inútil ejercicio académico; para obtener sorprendentes prestaciones del propio ordenador es necesario —a veces imprescindible, cuando se quiere ahorrar memoria y tiempo de ejecución— programar directamente el microprecesador que lo gobierna.

La CPU

El término CPU se usa muy a menudo con dos acepciones distintas. Considerando el ordenador dotado de periféricos, se designa a veces como CPU a la parte que contiene la unidad central propiamente dicha (aquélla que ejecuta las instrucciones en la secuencia correcta), la memoria central y los interfaces de conexión con el resto del ordenador (y, en muchos personales, también con el teclado). En este caso CPU o unidad central es sinónimo de: «el ordenador entero y verdadero, excluidos los periféricos físicamente separados de él (pantalla, grabadora, impresora, etc.)».
En cambio, desde el punto de vista lógico, la CPU es solamente la unidad central de cálculo y control, excluyendo, por tanto, memorias e interfaces de cualquier tipo. Dado que nosotros siempre habíamos entendido el término «CPU» (y también el término «unidad central») solamente en esta segunda acepción, lo seguiremos haciendo a lo largo de todos nuestros capítulos.
Hecha esta debida precisión, entremos rápidamente en el meollo del tema de esta lección, es decir, la descripción pormenorizada de la CPU.
Hasta ahora, siempre habíamos acudido a la unidad central como a una especie de «caja mágica», a la cual hacíamos preguntas en entrada (siguiendo una sintaxis y una gramática determinadas) y de la cual obteníamos en salida las respuestas.
Este es el momento preciso de levantar esta limitación, tratando de penetrar en ella con mayor profundidad. El conocimiento —más o menos profundo— de la unidad central no suele exigírsele generalmente al programador; aún más, en los límites de lo posible, todos los lenguajes de alto nivel (como el BASIC) tratan de evitar que existan ataduras de interdependencia entre el hardware y el software.
La fase de escritura de un programa (por lo menos en un lenguaje de alto nivel) debe, por tanto, ser siempre desarrollada con absoluta autonomía de las características hardware del ordenador utilizado, de tal forma que se resienta lo menos posible de las peculiaridades constructivas típicas de un sistema determinado.
El BASIC, como sabes bien —y sobre todo a causa de varios motivos

históricos que han provocado una evolución anómala— no puede ciertamente ser considerado en absoluto como el lenguaje más «transportable» y «compatible» (basta echar una ojeada a un idéntico programa escrito para dos ordenadores diferentes para darse cuenta de esto inmediatamente). Sin embargo, en los últimos años han aparecido muchos otros lenguajes de alto nivel, que permiten alcanzar el objetivo de la transportabilidad (es decir, de la casi perfecta compatibilidad entre máquinas distintas) casi completamente.

No obstante, en determinadas circunstancias, sobre todo en los casos de utilización especialmente «evolucionada» del ordenador o de programación en lenguajes de bajo nivel (es decir, mucho más próximos a la máquina que al hombre) puede ser, en cambio, de gran utilidad hacer referencia específica a un ordenador determinado y —en consecuencia— a un microprocesador específico. Además, a nivel de cultura general, el conocimiento de la constitución de la unidad central —por muy simplificada o limitada que sea— puede resultar un complemento necesario o incluso imprescindible.



HARDWARE

Organización externa de una CPU

Un microprocesador no sabe hacer nada solo. Para que pueda trabajar es necesario conectarle las memorias y los interfaces. Una vez conectado a estos circuitos el microprocesador se convierte en un microordenador, generalmente divisible en cuatro partes diferentes, que comunican unas con otras a través de un bus de datos, un bus de direcciones y un bus de control (por el momento, puedes asociar a la palabra «bus» la correspondiente castellana «conexión»).
Estas cuatro partes son:
1) El microprocesador.
2) La memoria de programa, que indica a la CPU las funciones a cumplir. Puede ser —según los casos— memoria ROM o memoria RAM.
3) La memoria de datos, que compagina los datos provenientes de los periféricos, los resultados intermedios y los resultados finales.
4) Los interfaces, que permiten la transferencia de los datos de los periféricos al microprocesador, y viceversa.
Examinemos un momento de cerca los tres últimos.
— La memoria de programa contiene la secuencia de instrucciones que el microprocesador debe ejecutar. Como ya hemos comentado, puede ser de tipo RAM o ROM. Cuando por ejemplo tú enciendes tu ordenador el rótulo que aparece en la pantalla está gestionado en la CPU por un programa que se encuentra en la ROM, mientras que cuando escribes una instrucción mediante el teclado ésta se memoriza en la RAM.
— La memoria de datos es, en cambio, un tipo de memoria notablemente distinta de la memoria de programa. Es a la fuerza una memoria RAM, dado que tiene que poder ser leída o escrita todas la veces que lo desees. Su presencia es indispensable, ya que el microprocesador —para poder ejecutar su programa— debe utilizar datos que, necesariamente, estén contenidos en la memoria de datos o que provengan de los periféricos a través de interfaces.
— Los interfaces, por último, como ya sabes bien, son circuitos de entrada-salida cuya función está definida por un programa. Permiten la comunicación entre el ordenador y los periféricos.

Organización interna de una CPU

Un microprocesador contiene bastantes miles de transistores y de otros componentes electrónicos: no es este el lugar de describirlo con grandes detalles (y por otra parte, esto tampoco seria de gran utilidad).
En cualquier caso, un microprocesador está constituido por tres partes principales:
- la unidad de control
- la unidad aritmético-lógica
- los registros.

La unidad de control decodifica las instrucciones que son tomadas de la memoria de programa y elabora las señales de comando necesarias para la ejecución de una instrucción.
La función de la unidad artimético-lógica (también llamada abreviadamente ALU) consiste, en cambio, en la ejecución de las operaciones lógicas y aritméticas sobre los datos que la alimentan a través de sus dos puertas de entrada, que son, respectivamente, «la entrada izquierda» y «la entrada derecha»: estas puertas se pueden imaginar como las dos extremidades más altas de un diagrama en forma de «V».
Después de la ejecución de una operación aritmética, como una suma o una resta, la ALU hace salir sus contenidos por la punta de la «V».
Los registros son de dos tipos: algunos son accesibles por programa, otros son internos de la CPU y no tienen gran importancia conceptual. Los registros accesibles se dividen en tres categorías:
- los registros de los datos
- los registros de las direcciones
- el contador de programa (Program Counter), el puntero del stack (pila) y el registro de estado.

Los registros de los datos son en la práctica localizaciones de memoria que permiten la memorización temporal de las informaciones durante sus desplazamientos entre la unidad aritmético-lógica, las memorias y los interfaces.
Su longitud, es decir, el número de bits que componen cada registro es obviamente igual a la de la palabra del microprocesador: 8 bits, si el microprocesador opera sobre 8 bits (como, por ejemplo, es el caso del Z80, es decir, de la CPU montada en el Spectrum).
Los registros de dirección, llamados también punteros, contienen direcciones de las posiciones de memoria que son enviadas al bus de dirección con una instrucción determinada que permita el acceso a estas posiciones.
¿Te acuerdas de

cuando –hablando de las memorias– comentabamos al respecto que cada localización disponía de una dirección bien precisa? Bien, esta dirección sirve a los registros de dirección para hacer que la CPU pueda referirse a cualquier posición de la memoria.
El puntero del stack es un registro de direccionamiento especial que señala a una determinada zona de la memoria, llamada «área del stack». Se decrementa automáticamente después de cada transferencia de una información a esta zona, y se incrementa después de cada remoción. El puntero del stack, como ya hemos podido comentar en una de las lecciones pasadas, tiene una función especialmente importante en la llamada y en el retorno de los subprogramas.
El Program Counter (o contador de las instrucciones) vigila la ejecución de todo el programa.
Inicialmente se carga con la dirección de la primera instrucción del programa, y, a continuación, señala al microprocesador las direcciones de las instrucciones que deben ser ejecutadas sucesivamente.
Mientras el microprocesador lee en la memoria de programa una instrucción, y la ejecuta, el contador prosigue hasta la dirección de la siguiente instrucción y así sucesivamente.
Generalmente el microprocesador ejecuta las instrucciones en orden secuencial, es decir, una detrás de otra. Sin embargo, en el caso de algunas instrucciones, la ejecución secuencial del programa puede ser modificada: en el Program Counter se carga entonces la dirección de salto, y se conserva la eventual dirección de retorno al programa principal para ser utilizada a continuación.
El registro de estado contiene en cambio un cierto número de bits puestos a 0 o a 1, según hayan sido verificadas determinadas condiciones después de la ejecución de algunas instrucciones (por ejemplo, si el resultado es positivo o negativo, con resto o sin él, etc.).
El desarrollo del programa puede ser modificado en función de los valores tomados por uno o más bits del registro de estado. Es uno de los registros más importantes que el programador tiene a su disposición. Los registros pueden ser utilizados para diversas comprobaciones especiales o condiciones excepcionales, o también para verificar velozmente algunos resultados equivocados.
Existe finalmente otro

importantísimo registro que equipa a la ALU: el acumulador. Este es siempre una de las dos entradas de la ALU (poco importa si es la «izquierda» o la «derecha»); la ALU hace siempre automáticamente referencia a este acumulador como a una de las entradas.
En las operaciones aritméticas y lógicas, por tanto, uno de los operandos estará en el acumulador y el otro se encontrará normalmente en una localización de la memoria.
El resultado será depositado en el acumulador. La referencia al acumulador como fuente y destino de los datos es la razón de su nombre: él acumula los resultados.
La ventaja de esta aproximación basada en el acumulador es la constituida por el hecho de que se pueden emplear instrucciones mucho más cortas en código máquina.
Si también el otro operando debiera ser tomado de uno de los otros registros (distintos del acumulador), sería necesario utilizar instrucciones más largas para indicar la dirección del registro.
Por esta razón, la arquitectura del acumulador se resuelve en una mayor velocidad de ejecución.
La desventaja es que el acumulador debe siempre ser cargado con los datos pedidos por la operación antes de su utilización.
Esto puede provocar en algunos casos algunas ineficiencias.

Los buses

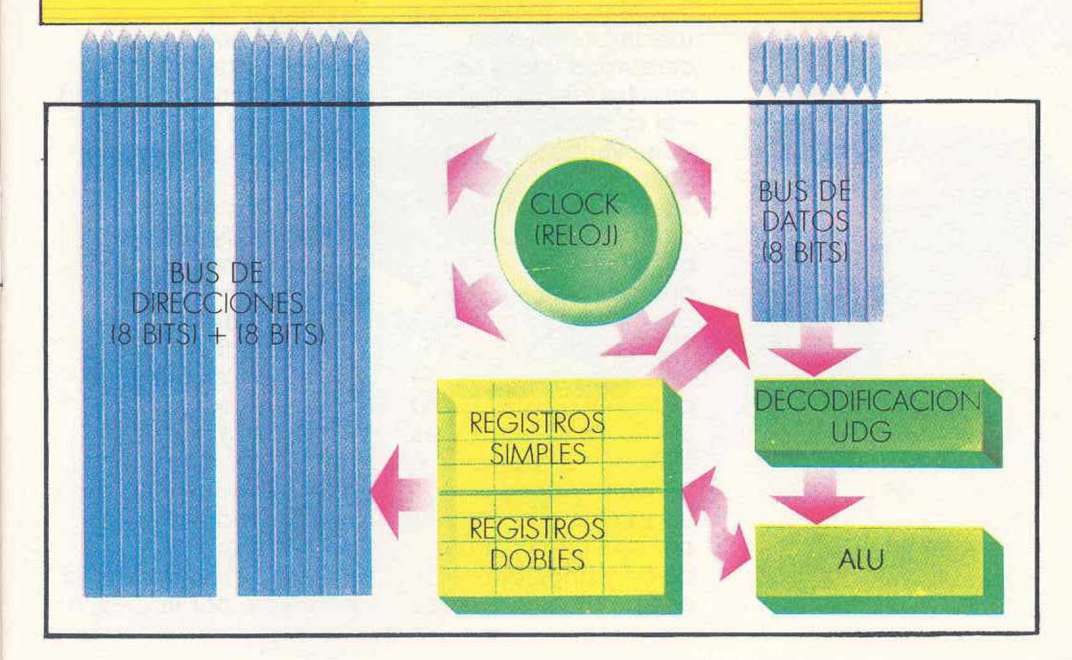
El microprocesador se comunica con los periféricos a través de tres buses:
– el bus de direcciones
– el bus de los datos
– el bus de control.

El bus de direcciones es un conjunto de líneas eléctricas, que permiten al microprocesador seleccionar una posición de memoria o un registro de un interface. La CPU envía sobre estas líneas, hacia un periférico, una dirección codificada en binario. El periférico, recibida y decodificada la dirección, selecciona el registro correspondiente.

El número de líneas del bus de direcciones determina lo que se llama potencia de direccionamiento del microprocesador; por ejemplo, 16 líneas permiten direccionar 2 ↑ 16 (65536) localizaciones de memoria. Es notable –entre otras cosas– que el concepto de dirección es muy similar al que se usa en el lenguaje corriente: la localización de una persona en una ciudad tiene lugar efectivamente a través de una dirección que contiene la calle y el número.

El bus de datos está también constituido por un grupo de líneas eléctricas en las cuales tiene lugar el intercambio de datos entre el microprocesador y los periféricos (memoria e interfaces). El número de líneas de este bus depende de la longitud de palabra del microprocesador: ya que el microprocesador del Spectrum es de 8

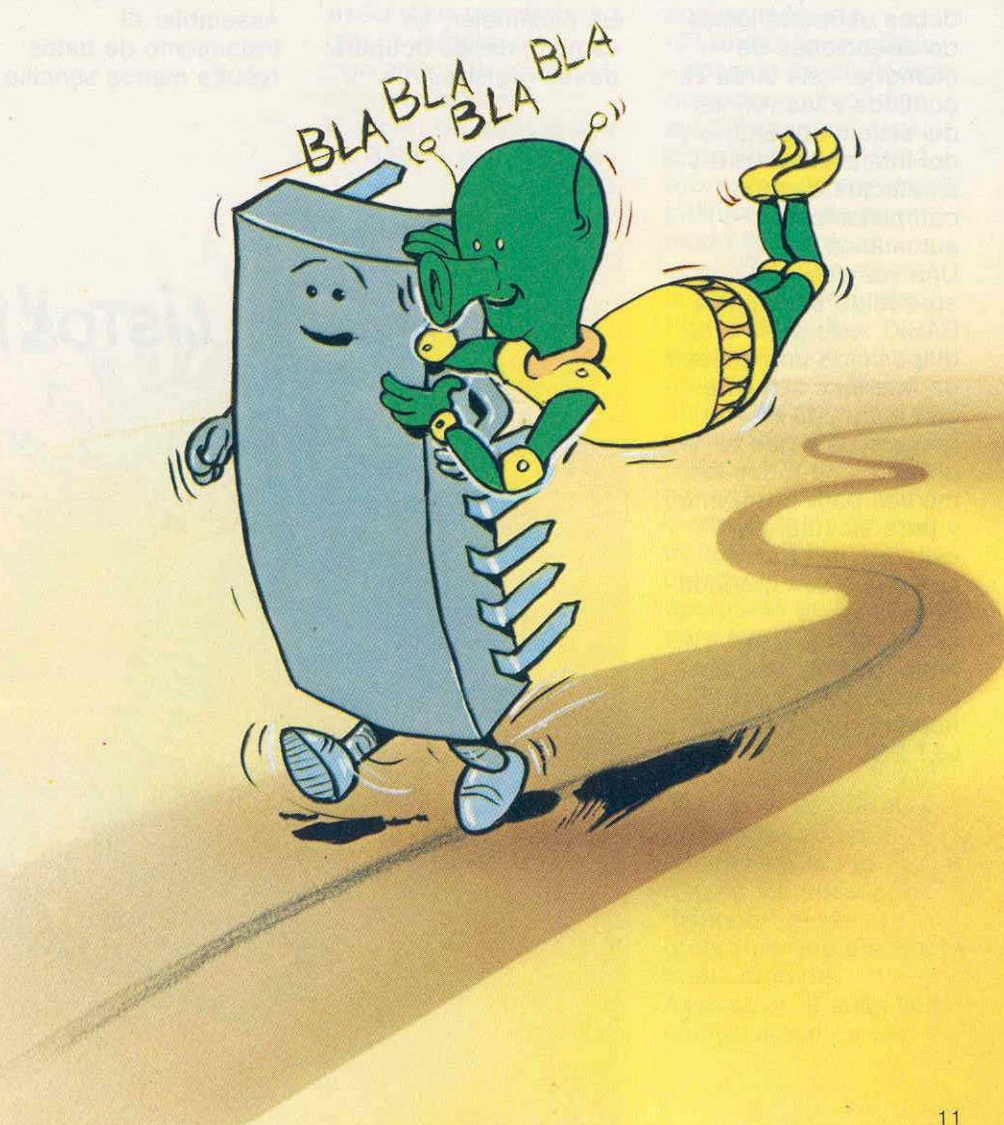
bits, el bus de datos dispone también de 8 líneas.
El bus de control está constituido por un cierto número de señales de diverso tipo, que aseguran la sincronización entre el microprocesador y los periféricos. Las funciones más comunes garantizadas por este bus son:
– la selección de una operación de lectura o escritura
– la interrupción del microprocesador (por interrupción se entiende un procedimiento que –a través de la señal eléctrica en una patilla especial de la CPU– indica al microprocesador que un periférico quiere comunicarse con él)
– la petición de acceso al bus de un periférico
– el reconocimiento de una petición de acceso al bus
– otras funciones menos comunes, pero tan importantes como las comentadas hasta ahora.



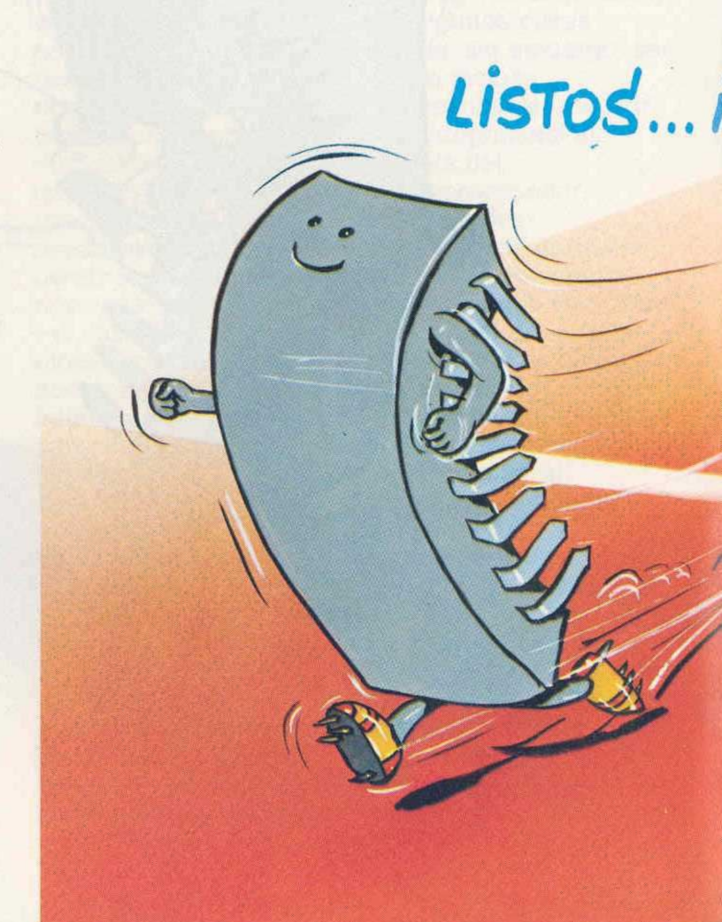
El código máquina

Hemos dicho ya varias veces que escribir un programa significa escribir una cierta serie de instrucciones en un lenguaje comprensible para el ordenador, es decir, en un lenguaje de programación.
Hagamos por un momento una analogía con el hombre.
Nosotros podemos hablar muchas lenguas (inglés, francés, alemán, etc.), podemos comprenderlas todas, pero si hacemos un razonamiento o pensamos un problema, nos damos cuenta que sólo empleamos una sola. Esta lengua, usada por nuestro cerebro, a veces es nuestra lengua materna —el castellano—, pero otras veces no es más que una «lengua interna». Algo muy parecido les sucede también a los ordenadores.
Ellos pueden, de hecho, comprender muchos lenguajes de programación, el BASIC, el FORTRAN, el COBOL, el PASCAL, etc., pero en su interior usan solamente uno, formado por una larga fila de ceros y unos en código binario. Este es el lenguaje de la máquina. Cualquier instrucción —por ejemplo, en BASIC— debe ser traducida a código máquina para poder ser ejecutada por el ordenador. Esta misión es realizada en el BASIC de tu Spectrum —ya lo sabes muy bien— por el incansable intérprete BASIC, auténtico traductor simultáneo entre tú y tu ordenador.
En algunos casos puede, sin embargo, ser útil (o incluso indispensable) recurrir a la programación directa del microprocesador, salvando los mecanismos normales (que inevitablemente disminuyen la velocidad de cálculo de la CPU) impuestos por el intérprete.
Programar en código máquina significa proporcionar al ordenador un cierto conjunto de configuraciones binarias (llamadas también códigos operativos) que el ordenador es capaz de comprender y ejecutar; cada una de ellas representa una operación ejecutada vía hardware por la CPU, es

decir, mediante las propias conmutaciones de interruptores en el interior del ordenador. La diferencia fundamental es que para programar en BASIC no es necesario conocer como funciona el microprocesador con el cual se ha realizado el ordenador, mientras que esto sí es necesario para programar en Assembler (así se llama

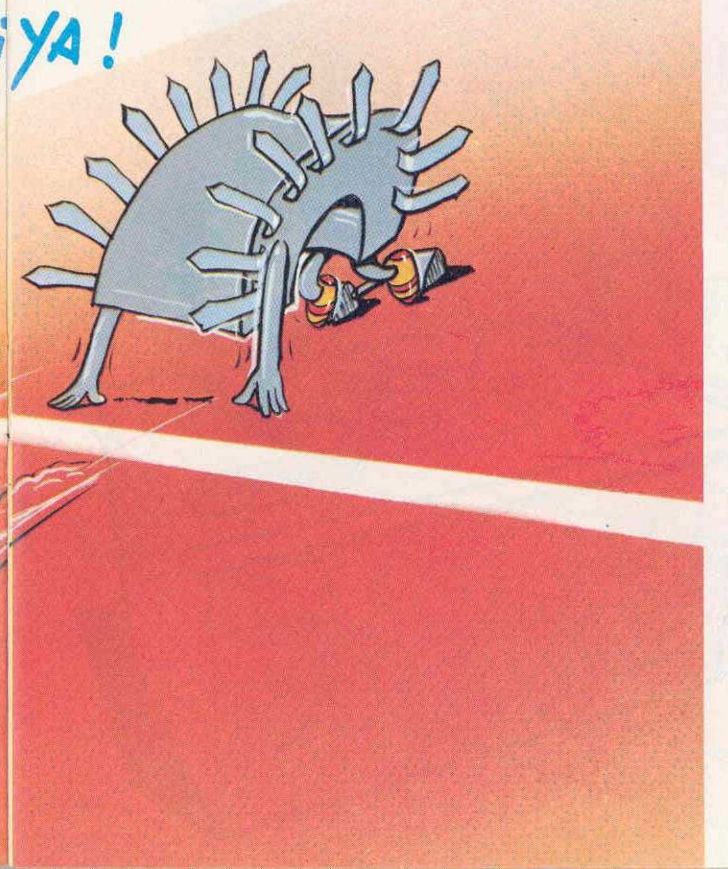


normalmente al código máquina).
En BASIC, salvo que se usen las instrucciones que leen y escriben directamente en los bytes de memoria (es decir, PEEK y POKE) no debes ocuparte jamás de direcciones de memoria: esta tarea es confiada a las rutinas del sistema operativo y del intérprete BASIC y se efectúa de modo completamente automático.
Una vez que has aprendido el lenguaje BASIC, este pone a tu disposición una especie de interface software con el cual tú te comunicas y que te proporciona todos los medios para programar y para ejecutar con éxito tus programas.
Los datos son tratados con nombres simbólicos (es, por tanto, posible asignar un número a cada nombre: por ejemplo LET PRECIO = 5000); las referencias a puntos específicos del programa (GOTO, GOSUB) se obtienen con referencias a los números indicadores de las líneas del programa BASIC.
En Assembler, en cambio, debes ocuparte de los registros funcionales del microprocesador, de las direcciones de memoria y cada operación debe ser pensada en todos sus detalles, siguiendo las características del ordenador. Además, en Assembler el tratamiento de datos resulta menos sencillo.



Desventajas y ventajas del Assembler

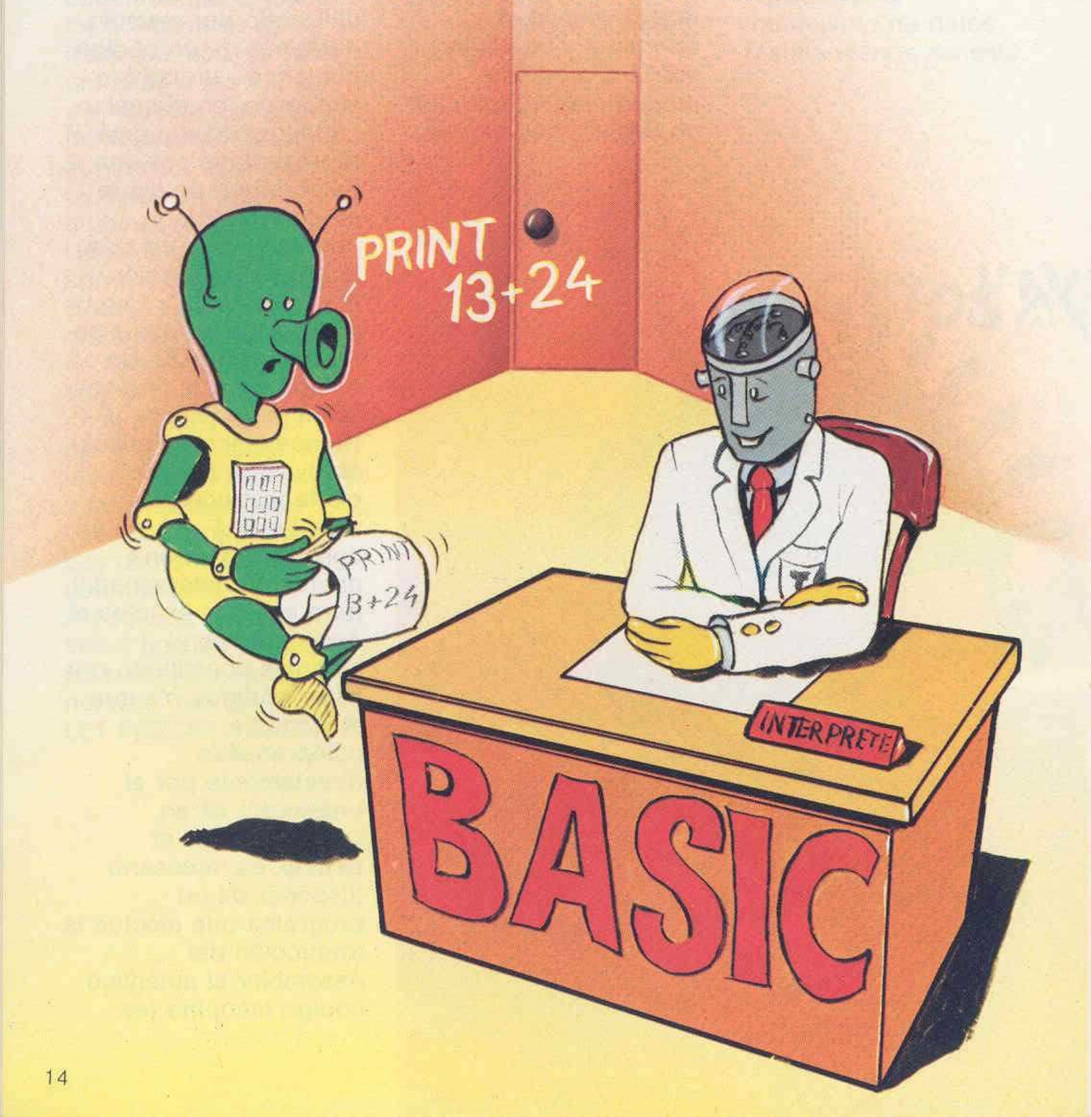
El código máquina está en binario: puedes intuir que escribir un programa en este lenguaje es largo, trabajoso y difícil. Por otra parte los programas en Assembler permiten una mayor velocidad operativa y un control sobre la máquina mucho más directo que en BASIC, y en ciertos casos es por esto necesario o conveniente someterse a este trabajo, cuando la aplicación lo requiere. Existen diversas posibilidades de escribir programas en código máquina, utilizando por ejemplo —además de un código numérico— un código simbólico, en el cual cada instrucción está representada por una palabra que de algún modo recuerda la operación que ejecuta la propia instrucción. Por ejemplo, la instrucción de suma se resume con ADD. Un código de este tipo se llama mnemónico («mnemónico» significa abreviatura de una cierta instrucción, que para la CPU corresponde a una operación determinada); para ser más exactos el Assembler es el lenguaje constituido por estos códigos. Ya que el lenguaje comprensible directamente por el ordenador es en cualquier caso el binario, es necesario disponer de un programa que efectúe la traducción del Assembler al auténtico código máquina (es



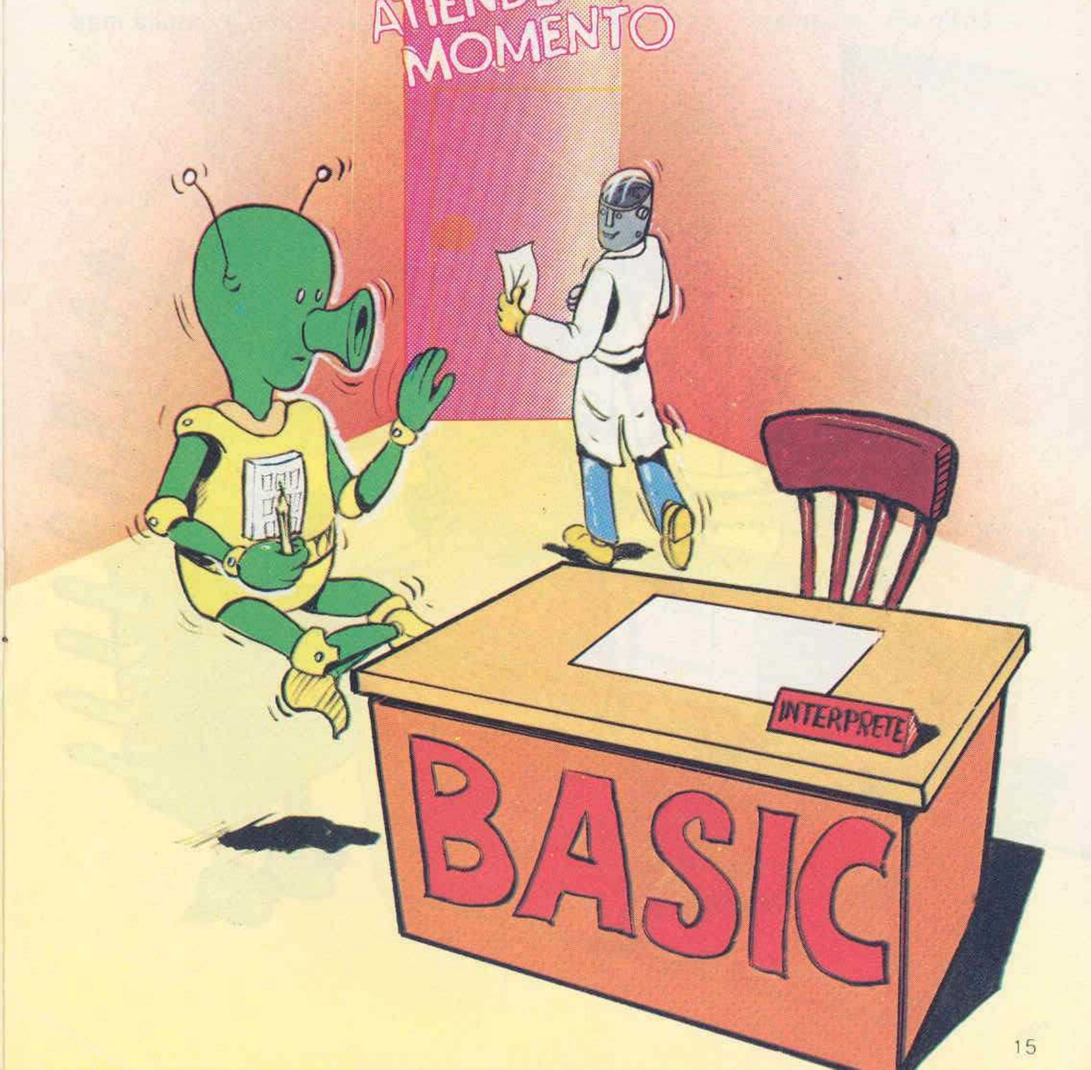
decir, que sustituya el código mnemónico de una determinada instrucción por el correspondiente código numérico); este programa recibe el nombre de ensamblador.

Existen muchos ensambladores en el mercado: su uso para quien desee programar seriamente en Assembler, es prácticamente indispensable. Teniendo en cuenta el carácter de introducción al código máquina de esta y de las próximas lecciones, no es de ninguna forma absolutamente necesario que te apresures en procurarte uno.

Para terminar,



indicaremos cuáles son las situaciones principales que pueden justificar el empleo del Assembler:
– todas las ocasiones en las cuales el factor tiempo juega un papel fundamental. Entre el tiempo de ejecución de un programa que funciona bajo el intérprete BASIC y el de un programa escrito directamente en Assembler puede existir una diferencia de velocidad enorme (el código máquina puede ser hasta 2-300 veces más rápido);
– todas las ocasiones en las cuales el factor espacio tiene una importancia esencial. La dimensión de un programa BASIC,

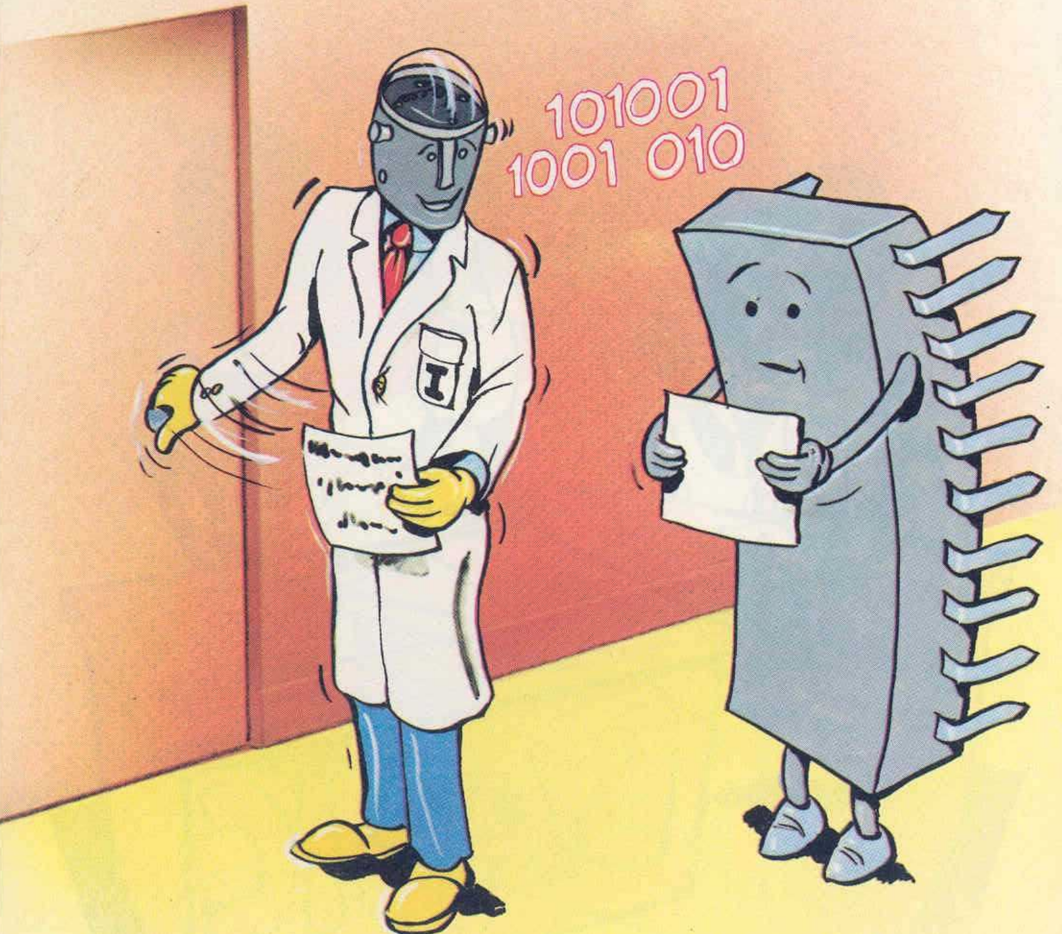


añadida a la de su intérprete, será siempre superior a la del mismo programa (que ejecuta la misma función) realizado en Assembler.

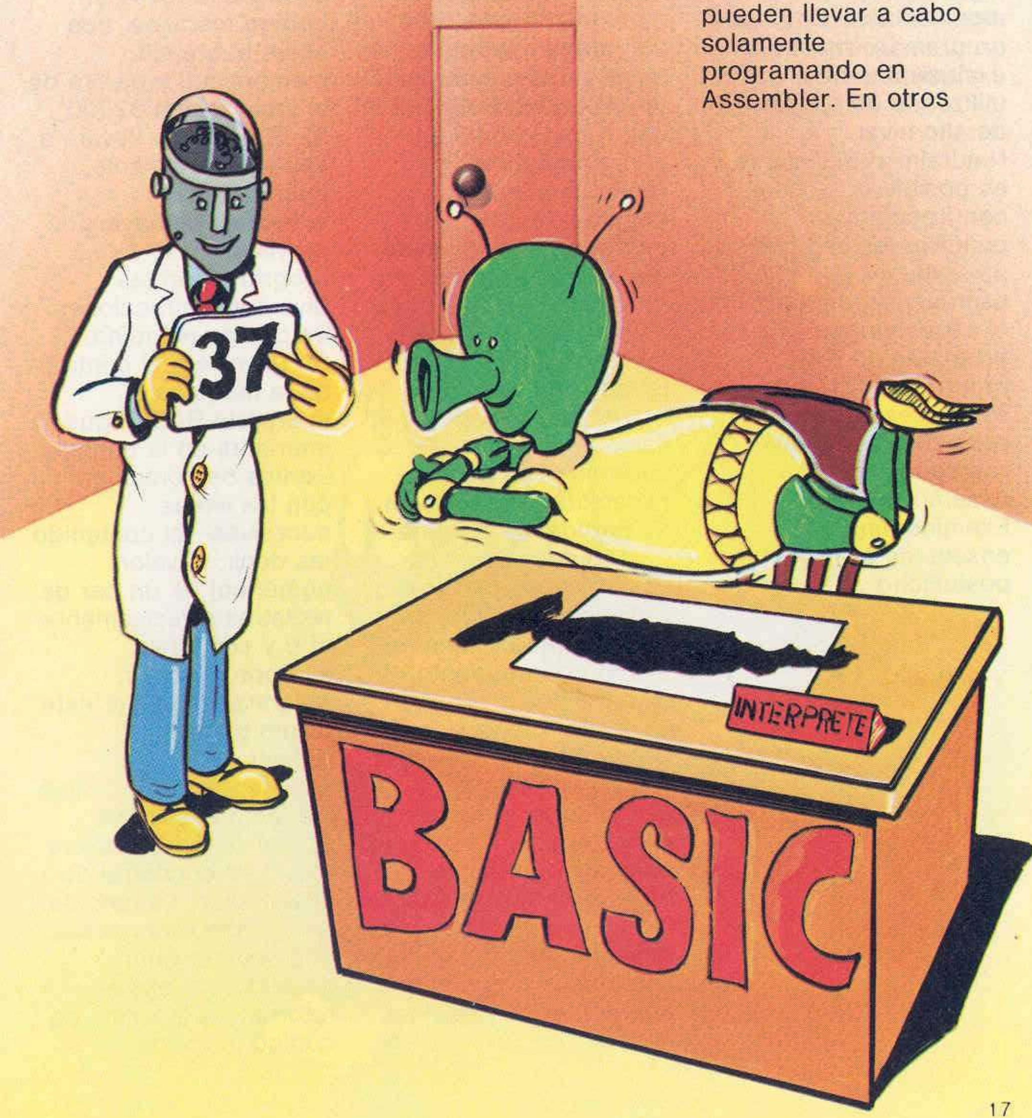
Este factor, con la progresiva e imparable caida de los precios de las memorias, está convirtiéndose en cualquier caso en el menos importante;

– todas las ocasiones en las cuales es necesario realizar instrucciones que son desconocidas para el intérprete BASIC, o imposibles de realizar con un lenguaje de alto nivel (como por ejemplo el BASIC).

Dado que un programa en Assembler resulta mucho más rápido que un programa en BASIC, pero requiere mucho más tiempo para escribirlo y resulta más



difícil encontrar los errores, será siempre necesario tomar de vez en cuando una decisión acerca del camino que conviene elegir, valorando bien las diversas exigencias, las posibles alternativas y los respectivos costes (no sólo en términos de tiempo, sino también en los de trabajo). Además, tienes que tener presente que ciertas aplicaciones particulares, como la utilización del ordenador para juegos de animación, se pueden llevar a cabo solamente programando en Assembler. En otros



casos (ejemplo típico: los programas de gestión financiera o de contabilidad) se requieren en cambio fundamentalmente facilidades de legibilidad y modificabilidad de los programas, obtenibles exclusivamente utilizando un lenguaje de alto nivel.
Naturalmente, siempre es posible (y se hace con frecuencia) conciliar las dos formas de lenguaje, escribiendo por ejemplo el esqueleto del programa en BASIC y recurriendo, cuando sea necesario, a pequeños programas realizados en Assembler.
Examinaremos enseguida esta posibilidad.

USR

USR es un comando BASIC que significa «User SubRoutine» (rutina definida por el usuario). Resulta muy útil cuando se desea pasar directamente de un programa BASIC a una rutina escrita en código máquina.
Naturalmente, es necesario que en el momento de la llamada de USR el programa en código máquina esté ya memorizado en el interior del Spectrum.
Es además necesario —igual que sucede en la llamada de las subrutinas en BASIC— especificar la dirección de partida de la rutina que se desea ejecutar.
Para ser más precisos, USR es por tanto una función: requiere en efecto un argumento bien difinido, correspondiente a la localización de memoria en la cual deberá comenzar la ejecución del programa en código máquina, y proporciona un resultado según las reglas que veremos más adelante. Así, si escribimos un programa que contenga esta línea BASIC:

```
100 PRINT USR 32000
```

nuestro Spectrum ejecutará en orden esta serie de operaciones:
1) Pasará el control del BASIC a la rutina en código máquina, que antes habrá sido memorizada y partirá de la localización 32000.
2) Ejecutará y llevará a cabo (naturalmente cuando esta eventualidad haya sido prevista por el programador) las diversas instrucciones en código máquina.
3) Devolverá el control de la máquina al intérprete BASIC, que imprimirá en la pantalla —antes de continuar con las líneas sucesivas— el contenido (es decir, el valor numérico) de un par de registros (precisamente el B y el C) del microprocesador.
Aclararemos mejor este último punto.
Al final del subprograma en código máquina el sistema operativo del Spectrum recupera el control de la situación. La función USR, entre otras cosas, hace que el valor asumido por ella al retornar de la rutina en código máquina

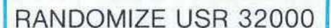
(acuérdate que las funciones en salida proporcionan siempre un resultado) sea igual al valor contenido en el par de registros BC del Z80 (es decir, de la CPU del Spectrum). El microprocesador dispone de numerosos registros –utilizados para efectuar los diversos cálculos– convencionalmente designados con algunas letras del alfabeto. Entre estos están el B y el C. Si hubiésemos escrito:

```
100 LET A=USR 32000
```

al finalizar la rutina en código máquina el valor contenido en BC hubiera sido en cambio memorizado en la



variable A.
En algunas circunstancias, sin embargo, visualizar sobre la pantalla o memorizar en una variable numérica el contenido de la pareja BC puede no ser de ninguna utilidad práctica. En este caso se puede hacer uso de un



```
RANDOMIZE USR 32000
```

(siempre suponiendo que hayamos memorizado la rutina en

código máquina a partir de la localización 32000).
Recurriendo a esta posibilidad todo sucede de una forma absolutamente idéntica a como hemos visto antes; la única diferencia reside en el hecho de que ahora el resultado de la función USR alterará la variable de control usada para generar los diversos valores de RND. Este hecho puede ser aprovechado útilmente cuando, por ejemplo, se desea proyectar un programa que mueva objetos en pantalla de una manera aparentemente al azar.
Si en cambio la generación de los números no es de ningún interés, tampoco importa: lo importante es que la rutina en código máquina haya sido ejecutada y que el retorno al BASIC tenga lugar sin «ensuciar» la pantalla o ocupar memoria.
Aunque los ejemplos que examinaremos más adelante (en la parte de la lección dedicada a la programación) esclarecerán este hecho, es importante recordar que con USR el ordenador ejecuta una auténtica llamada a una subrutina (en este caso, escrita –lo repetimos– en código máquina). En otras palabras, después de haber hecho ejecutar al Spectrum una instrucción del tipo PRINT USR... o RANDOMIZE USR... es necesario insertar una instrucción en código máquina, que le diga al ordenador que debe volver al BASIC. Es lo mismo que sucede cuando se usa una instrucción GOSUB; para volver al programa principal es necesario insertar una instrucción RETURN al terminar el subprograma.
Esta instrucción en el Assembler del Z80, tiene como código el mnemónico RET y como código operativo el valor decimal 201.

Sintaxis de la función

USR dirección

Descubriendo el Assembler

Antes de pasar directamente a la escritura de programas en código máquina, debemos todavía afrontar otros temas importantes (es más, imprescindibles), gracias a los cuales podremos avanzar de una manera más fácil y sencilla.
Te podrá quizá parecer que el código máquina no es para tí: demasiadas cosas que aprender, que saber y que tener en la cabeza. Esto también puede ser cierto: sin embargo, una vez aprendidos los conceptos fundamentales, muchos temas te resultarán mucho menos complicados de lo que podría parecer a primera vista.
Además, cuanto más te adentres en el Assembler —haciéndote en consecuencia más dueño de la situación— más espectaculares y satisfactorios serán los resultados.

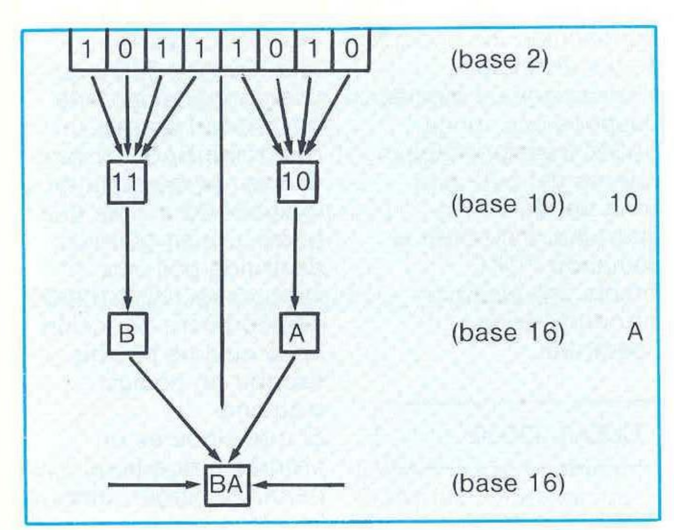
La numeración hexadecimal

El primer e importante punto es determinar un sistema de numeración que llegue a un compromiso entre la dificultad que el hombre encuentra en leer y trabajar con los números binarios y la aversión que la CPU posee hacia las cifras decimales.
Por esta razón, interviene un tercer y fundamental (para quien programa en Assembler) sistema de numeración, el de base 16. Con esta base un número binario de ocho cifras puede ser escrito con dos cifras en base dieciséis, es decir, con dos números hexadecimales.
Seguramente recordarás que un número en base dos está compuesto solamente por las cifras 0 y 1, mientras que uno en base 10 se escribe con cifras comprendidas entre 0 y 9; así, un número en base 16 deberá ser escrito con cifras comprendidas entre 0 y 15. Sin embargo, como no tenemos cifras (simples) mayores que 9, usamos las primeras seis letras del alfabeto:

10 = A
11 = B
12 = C
13 = D
14 = E
15 = F

El modo más sencillo de convertir un número binario de 8 bits en un número hexadecimal es el que consiste en separar los 8 bits en dos grupos de 4 bits, efectuar la conversión de cada grupo de una base a la otra y después combinar el resultado.

Por ejemplo:



Con este método podemos convertir cualquier valor de un byte en dos cifras hexadecimales. La ventaja consiste en obtener una escritura más compacta de la notación binaria y, con un mínimo de entrenamiento, casi idéntica al sistema decimal. En cualquier caso, todos estos sistemas de numeración son —recordémoslo— modos diferentes de representar los mismo números. El único secreto es conocer la base utilizada. Por convención, a fin de reconocer inmediatamente un número hexadecimal, escribiremos siempre los números en base 16 con el sufijo H. Así

```
10
```

significará «diez decimal», mientras que

```
10H
```

significará «diez hexadecimal» (es decir, 16 decimal).

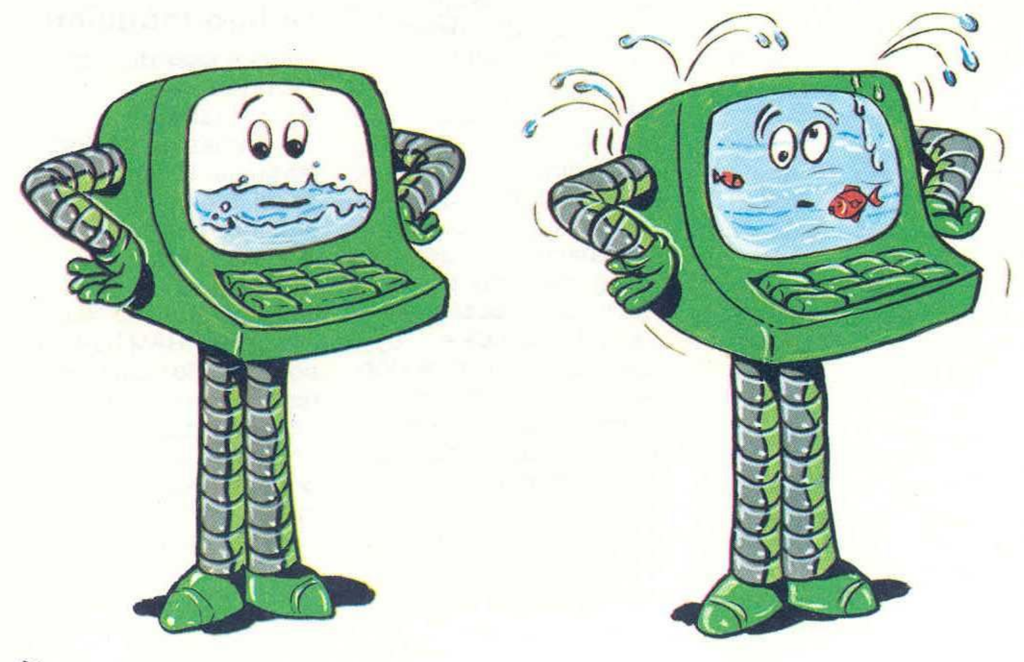
Cómo memorizar los programas en código máquina

Puedes usar distintos métodos para memorizar los programas en código máquina. El más sencillo es el que consiste en rebajar el TOP de la memoria dedicada al programa BASIC, actuando sobre el puntero RAMTOP. Si necesitas los caracteres gráficos de usuario bastará rebajar el contenido de RAMTOP de tal forma que no invada la zona de los caracteres gráficos de

usuario.
El rebajamiento del TOP de la memoria se obtiene con la instrucción CLEAR seguida por la dirección del último byte que quieres dejar disponible para el programa BASIC.
Las rutinas en código máquina pueden ser incorporadas en los programas BASIC, usando las instrucciones DATA seguidas por una serie de números que representan los códigos de las distintas instrucciones a ejecutar.
Después una rutina BASIC transferirá los valores del byte a la zona de memoria oportuna, utilizando el comando POKE.
Intenta por ejemplo introducir en tu Spectrum:

```
CLEAR 32000
```

Borrarás la memoria desde la localización 32000 hasta el final de la RAM (32767 en la versión 16K y 65535 en la versión 48K). Esta instrucción protegerá una eventual rutina en código máquina de la posibilidad de que un programa BASIC invada el área de memoria en la cual está escrita y la borre, o bien que sea destruída por una instrucción NEW. 32000 es la primera dirección en la cual es posible escribir en código máquina.
El que sigue es un programa que te permitirá cargar, una tras otra, las instrucciones de las rutinas en código máquina:



```
10 INPUT "RAMTOP="; RT
20 CLEAR RT
25 RESTORE
30 LET PRINCIPIO=1+(PEEK 23730+256*PEEK
   23732)
40 FOR A=PRINCIPIO TO 65535 (usa 32767 en
   la versión 16K)
50 READ X:IF X=999 THEN STOP
60 POKE A,X
70 NEXT A
80 DATA ...aquí se escriben los códigos...
```

Examinemos el programa línea por línea, para ver como funciona: la línea 10 pide la dirección de RAMTOP. La línea 20 fija el RAMTOP en la localización indicada. La 25 usa la instrucción RESTORE para posicionar el puntero data sobre el primero de los datos.
La línea 30 asigna a la variable PRINCIPIO la localización de partida del programa en CM.
No se puede hacer:

```
PRINCIPIO = RT + 1
```

porque la instrucción CLEAR borra también todas las variables: ya que las localizaciones 23730 y 23731 contienen la dirección del último byte disponible para el BASIC, bastará leer el contendio para establecer el valor correcto de PRINCIPIO.
La línea 40 comienza un bucle que va de la localización PRINCIPIO a la máxima dirección posible en el Spectrum (65535 en la versión de 48K y 32767 en la de 16K). La línea 50 lee los diversos elementos de las líneas DATA y comprueba cada vez si alguno es igual a 999 (que es un código no admitido para la instrucción POKE); de esta manera somos capaces de identificar el final de los datos. La línea 70 cierra el bucle. Naturalmente, no es necesario disponer todos los valores en una única línea DATA; puedes usar cuantas líneas desees, siempre que sean compatibles con la memoria disponible. La línea 80 (y las eventualmente siguientes) contiene los valores decimales de la rutina y debe tener como último elemento el 999, para que el ordenador pueda darse cuenta de que ha llegado hasta el final.
Así este programa carga el código máquina en la RAM.
Para hacer ejecutar la rutina será necesario modificar la línea 50, de tal forma que el programa no se pare al corresponderse con el 999, sino que envíe una línea PRINT USR PRINCIPIO o RANDOMIZE USR PRINCIPIO. Este ejercicio (en conjunto verdaderamente elemental) lo dejamos a tu cargo.
Como alternativa, puedes ordenar la ejecución impartiendo uno de los dos comandos USR en modo inmediato.

Ejemplos Assembler

Probemos ahora a escribir algunos programas muy sencillos en Assembler, mostrándote simultáneamente su funcionamiento y su comparación con el correspondiente listado BASIC. No siempre es posible hacer esta comparación: los ejemplos que trataremos ofrecerán, sin embargo, esta posibilidad.
Como primer ejemplo propongámonos transferir el contenido de dos localizaciones de memoria (por ejemplo la 5428 y la 5429) a otra pareja de localizaciones (elegimos la 25000 y la 25001).
En BASIC escribiríamos:





```
10 POKE 25000, PEEK 5428
20 POKE 25001, PEEK 5429
```

y todo estaría resuelto. Veamos ahora que hay que hacer en Assembler. He aquí el listado:

```
LD HL,(1534H)
LD (61A8H), HL
RET
```

La primera línea carga (LD es la abreviatura de LOAD, es decir, carga) el registro HL con el número contenido en la pareja de localizaciones situadas a partir de la dirección 5428 (1534 hexadecimal). La segunda memoriza en la pareja de localizaciones 61A8 y 61A9 (25000 y 25001 decimales) el valor contenido en HL. La tercera línea es el famoso RETURN, que sirve para devolver el control al BASIC.
Es interesante observar que en la práctica hemos trabajado sobre

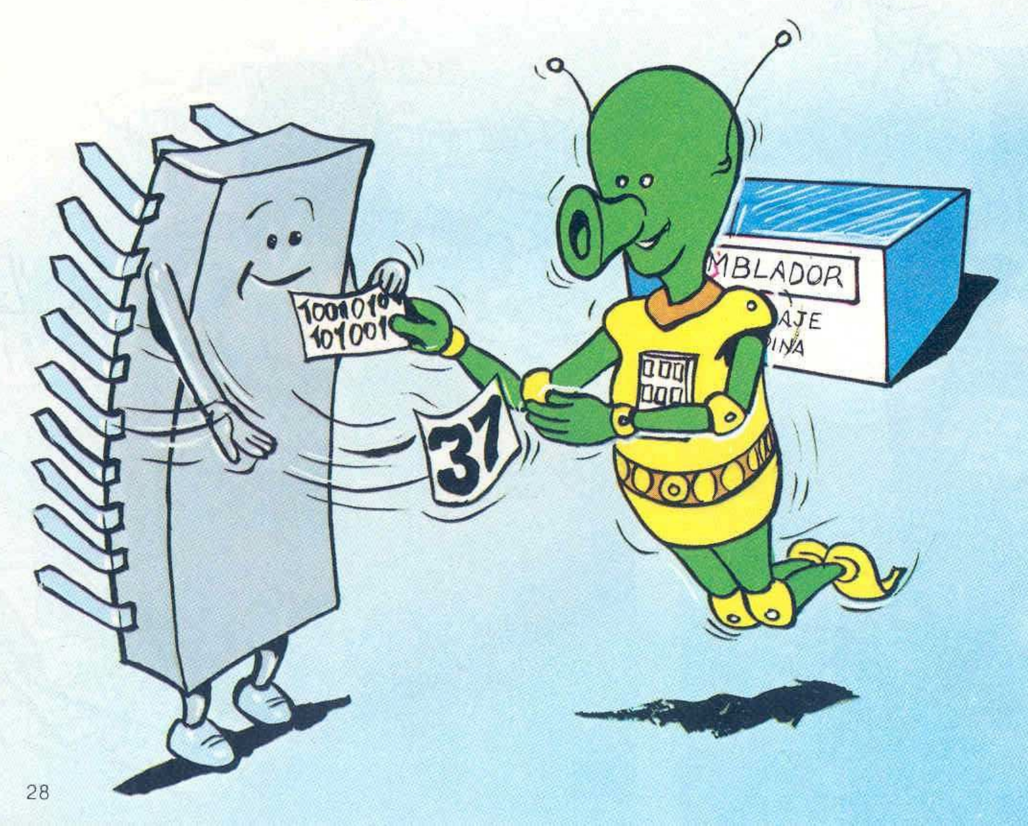




16 bits a la vez, transfiriéndolos primero a un registro doble (HL indica que hemos utilizado en el mismo momento los registros H y L) y después a dos localizaciones de 8 bits. Esto es posible gracias al diseño especial de los registros del Z80, que, incluso siendo de 8 bits, pueden ser compuestos de forma que se obtengan registros de 16 bits. Observemos ahora la constitución de las instrucciones: un operador y uno o más operandos. En LD HL,1534H –por ejemplo– LD es el operador, mientras que HL y 1534H son los operandos.
Es necesario hacer una precisión sobre la parte (o, mejor dicho, sobre el campo) relativa a los operandos.
Este campo depende estrechamente del tipo de operación y pertenece siempre a uno de los siguientes grupos:

– de un solo argumento
– de dos argumentos, separados por un carácter especial (generalmente una coma). En este último caso el primer argumento representa siempre el destino y el segundo la fuente.

Así, en LD HL,(1534H), la fuente (es decir, el número contenido en 1534H) debe ir a HL, que es el destino. Lo mismo ocurre en LD (61A8H),HL.
Pasemos ahora a la



conversión del programa de los códigos numéricos ejecutables (estos que acabamos de ver son solamente los mnemónicos). Es necesario tener en las manos la tabla proporcionada por el fabricante del microprocesdor, y convertir uno a uno los diversos términos. A cada elemento le corresponderá naturalmente un código numérico preciso:

```
LD HL,(1534H)        = 2A 34 15
LD (61A8H), HL       = 22 A8 61
RET                  = C9
```

Los números que hemos escrito están todavía en hexadecimal: antes de ser escritos en los DATA deberán ser convertidos en decimales.
La última cosa a tener en cuenta es que —si miras bien— las direcciones de las localizaciones han sido escritas invertidas (es decir 1534 ha sido escrito 3415, e igualmente ha ocurrido con 61A8).
Esto es debido a razones constructivas del microprocesador, el cual pretende que le sean proporcionados antes el byte bajo y después el alto de cualquier dirección.
Al final obtendremos los valores siguientes:

```
2AH = 42
34H = 52
15H = 21
22H = 34
A8H = 168
61H = 97
C9H = 201
```

que, insertados en la línea DATA (acordándose de añadir también el 999), formarán la rutina.
Ejecuta el programa y envía por tanto una USR. Intenta después comprobar con PEEK en las localizaciones 25000 y 25001 que el trabajo haya sido verdaderamente llevado a cabo.

Probemos ahora este segundo ejemplo:

```
LD B,00H    ;carga 00H en el registro B
LD C,00H    ;carga 00H en el registro C
LD A,0CH    ;carga 0CH en el registro A
SUB A,06H   ;resta 06H de A
LD C,A      ;carga el valor de A en C
RET         ;vuelve al BASIC.
```

Observa que en Assembler los comentarios se separan de las instrucciones mediante un punto y coma.
El equivalente BASIC del programa recién visto sería:

```
10 LET B = 0
20 LET C = 0
30 LET A = 12
40 LET A = A – 6
50 LET C = A
60 RETURN
```

La conversión del programa en los códigos numéricos lleva a los siguientes valores:

```
6, 0, 14, 0, 62, 12, 214, 6, 79, 201,
```

Si ejecutas esta rutina mediante un PRINT USR PRINCIPIO, verás que en la pantalla aparecerá —como ya habíamos dicho antes— el contenido de los registros B y C. Estos valdrán respectivamente 0 y 6.

Conversión de un carácter en CM

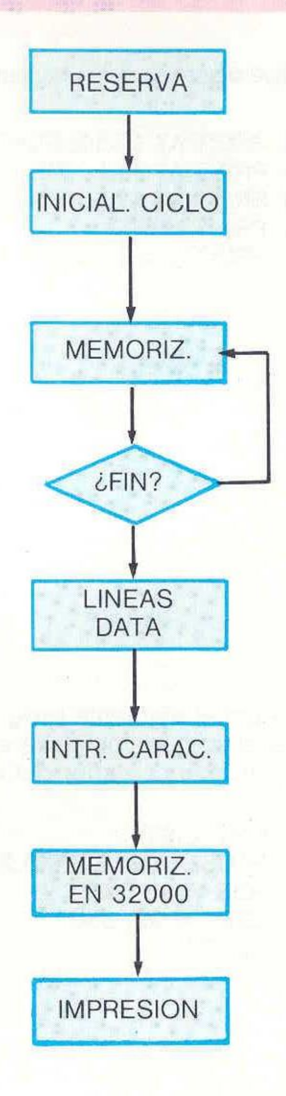
Al margen de la posibilidad de utilización real, el objetivo más importante de esta propuesta es el de subrayar el procedimiento correcto a seguir para plantear y ejecutar un programa cualquiera en código máquina.
Para convertir, así, un carácter minúsculo en el correspondiente mayúsculo, puedes usar muy bien el BASIC, pero este ejemplo ofrece la oportunidad de poner de manifiesto las tres fases fundamentales necesarias para introducir y utilizar rutinas en CM en tus programas.

PROGRAMACION

1 **Preparación:**
Se fija el RAMTOP para que el programa BASIC no pueda de ninguna manera superponerse al código máquina.

2 **Memorización:**
Se introduce en memoria el código máquina.

3 **Ejecución:**
Se hace correr el programa en CM.

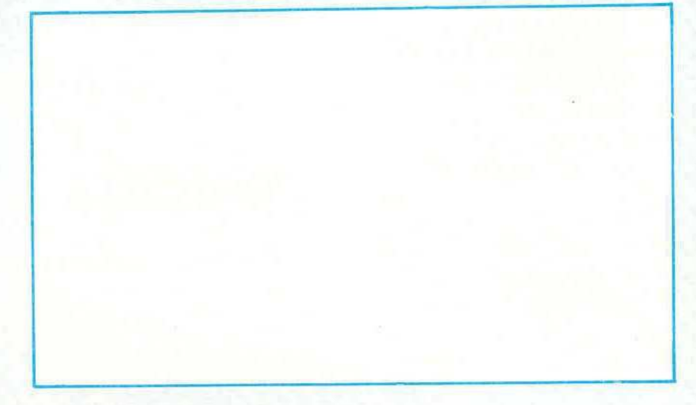



```
10 CLEAR 31999
20 FOR I=32001 TO 32009
30 READ X
40 POKE I,X
50 NEXT I
60 DATA 58,0,125,203,175,79,6,0,201
70 INPUT C$
80 POKE 32000, CODE C$
90 PRINT CHR$ USR 32001
```

EJERCICIOS

¿Qué aparecerá en la pantalla al final de la ejecución de este programa?

```
10 PRINT AT 10,13; FLASH 1; "PRUEBA"
20 PRINT AT 11,6; "DE UNA EXTRAÑA RUTINA"
30 PRINT AT 12,6; "EN CODIGO MAQUINA"
40 PRINT USR 0
50 PRINT "¡SI PASA DE AQUI ES UN MILAGRO!"
```



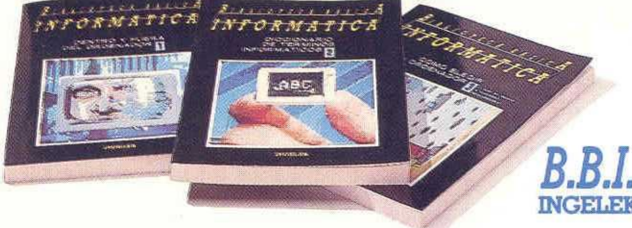
Ejecuta el siguiente programa y, a través de él, la rutina del sistema operativo que comienza en la localización 3582. Descubre tú mismo su efecto variando el contador del ciclo automático.

```
10 LIST: LIST
20 PAUSE 200: BEEP 1,20
30 FOR V= 1 TO 7
40 LET A=USR 3582
50 NEXT V
60 REM MIRA ARRIBA
```

UNA GRAN OBRA A SU ALCANCE



UNA OBRA COMPLETISIMA EN 30 VOLUMENES QUE TRATA TODOS LOS TEMAS , DESDE QUE ES UN ORDENADOR HASTA EL ESTUDIO DE LOS DIVERSOS LENGUAJES, PASANDO POR LOS LENGUAJES, METODOS DE PROGRAMACION, ELECCION DEL ORDENADOR ADECUADO, DICCIONARIO, ETC.



B.B.I.
INGELEK

30 EXTRAORDINARIOS VOLUMENES DE APARICION SEMANAL CON TODOS LOS CONCEPTOS DE LA INFORMATICA

GRAN OFERTA DE SUSCRIPCION
9.995 PTAS

AHORRE MAS DE 1.000 PTAS Y LLEVESE UNA MAGNIFICA CALCULADORA SOLAR VALORADA EN 2.500 PTAS.



OFERTA VALIDA UNICAMENTE PARA ESPAÑA

SUSCRIBASE POR TELEFONO

Todos los días, excepto sábados y festivos, de 8 a 6,30 atenderemos sus consultas en el

